Qt — графический фреймворк Qt для языка программирования С++. Qt Creator — кроссплатформенная свободная среда для разработки графических интерфейсов (GUI) программ использующих библиотеку Qt. Входит в состав Qt framework. Qt является фундаментом популярной рабочей среды KDE, входящей в состав многих дистрибутивов Linux.

Мы разработаем игру «Жизнь» Джона Конвея. Игра «Жизнь» — это клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году. Место действия этой игры — «вселенная» — это размеченная на клетки поверхность или плоскость — безграничная, ограниченная, или замкнутая. Каждая клетка на этой поверхности может находиться в двух состояниях: быть «живой» (закрашенной) или быть «мёртвой» (незакрашенной). Клетка имеет восемь соседей, окружающих её.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | :-) |  |
|  |  |  |

Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:

* в пустой (мёртвой) клетке, рядом с которой ровно три живые клетки, зарождается жизнь;
* если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить;
* в противном случае, если соседей меньше двух или больше трёх, клетка умирает («от одиночества» или «от перенаселённости»).

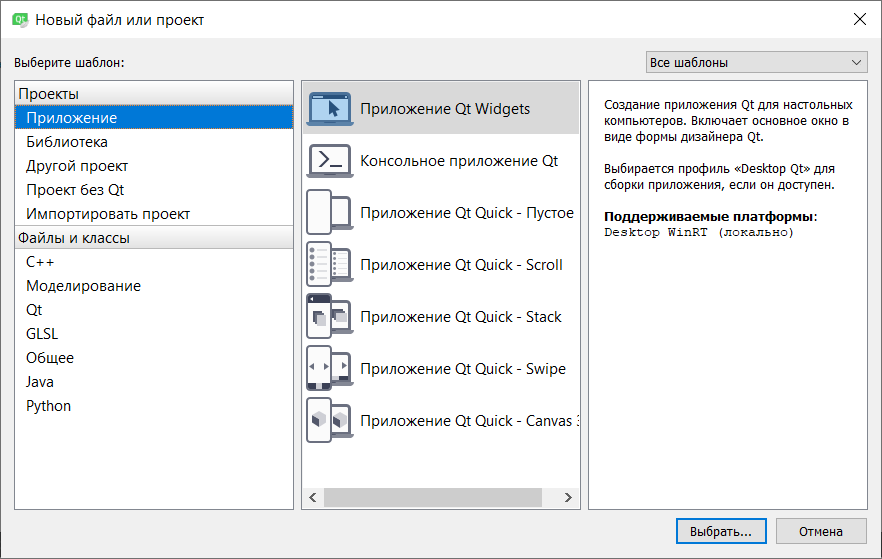
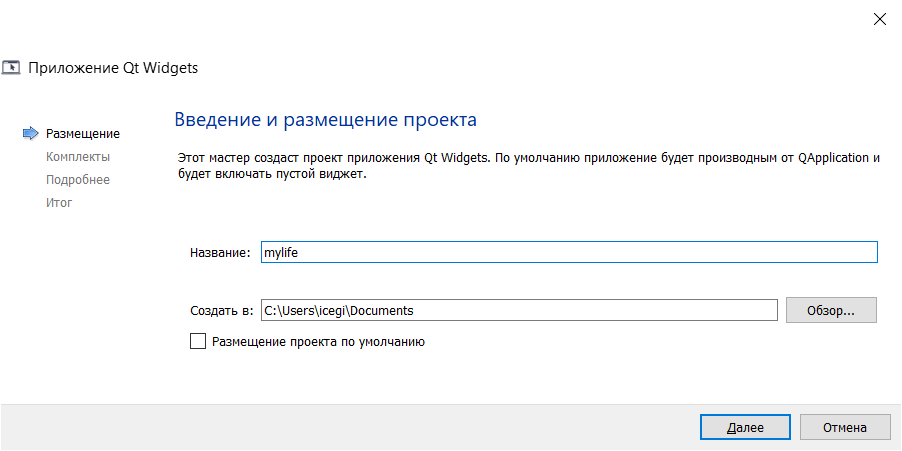
Игра прекращается, если

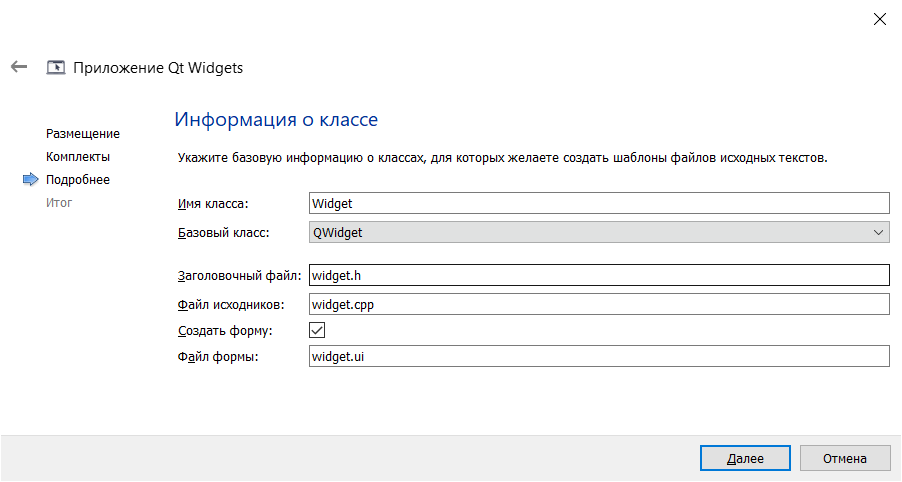
* на поле не останется ни одной «живой» клетки;
* конфигурация на очередном шаге в точности (без сдвигов и поворотов) повторит себя же на одном из более ранних шагов (складывается периодическая конфигурация);
* при очередном шаге ни одна из клеток не меняет своего состояния (складывается стабильная конфигурация; предыдущее правило, вырожденное до одного шага назад).

Эти простые правила приводят к огромному разнообразию форм, которые могут возникнуть в игре. Игрок не принимает прямого участия в игре, а лишь расставляет или генерирует начальную конфигурацию «живых» клеток, которые затем взаимодействуют согласно правилам уже без его участия (он является наблюдателем).

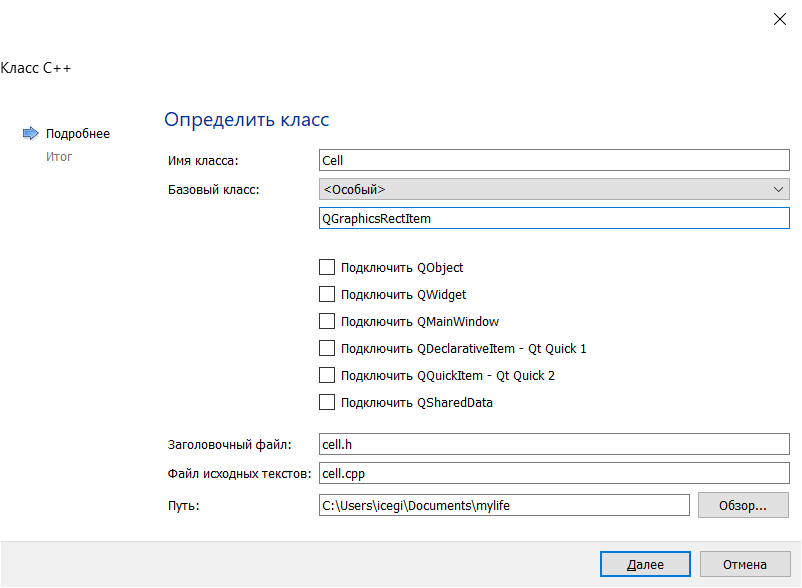
1. Открываем Qt Creator и создаем новый проект. Файл -> Создать файл или проект…

Приложение -> Приложение Qt Widgets -> Вводим название и путь к проекту -> Оставляем настройки по умолчанию -> Базовый класс QWidget -> Оставляем настройки по умолчанию -> Завершить



1. Создадим класс Cell, который моделирует клетку. Файл -> Создать файл или проект… Файлы и классы -> С++ -> Класс C++. Будем создавать Cell производным от класса QGraphicsRectItem, который описывает графический объект – прямоугольник.



Подключим в заголовочный файл класса Cell (cell.h) необходимые библиотеки.

#ifndef CELL\_H

#define CELL\_H

#include <QObject>

#include <QtWidgets>

class **Cell** : public QGraphicsRectItem

{

public:

**Cell**();

};

#endif

1. В конструктор Cell (метод создания объекта) будем передавать 4 параметра: координаты (x и у) и размер (ширина и высота).

cell.h

class **Cell** : public QGraphicsRectItem

{

public:

**Cell**(qreal x, qreal y, qreal w, qreal h);

};

cell.cpp

#include "cell.h"

Cell::**Cell**(qreal x, qreal y, qreal w, qreal h):

QGraphicsRectItem (x,y,w,h)

{

this->setBrush(QColor(255, 255, 255));

}

1. Переопределим обработчик события нажатия на кнопку мыши: клик левой кнопкой – закрашиваем квадратик в синий цвет, а правой – в белый. Также добавим метод changeColor(), меняющий цвет на противоположный. Закрашенные синим квадратики – «живые» клетки, а белым – «мертвые».

cell.h

class **Cell** : public QGraphicsRectItem

{

public:

**Cell**(qreal x, qreal y, qreal w, qreal h);

void **changeColor**();

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event);

};

cell.cpp

void Cell::**changeColor**()

{

if (this->brush().color() == QColor(255, 255, 255))

this->setBrush(QColor(0,64,128));

else

this->setBrush(QColor(255,255,255));

}

void Cell::***mousePressEvent***(QGraphicsSceneMouseEvent \*event)

{

if (event->button() == Qt::LeftButton )

this->setBrush(QColor(0, 64, 128));

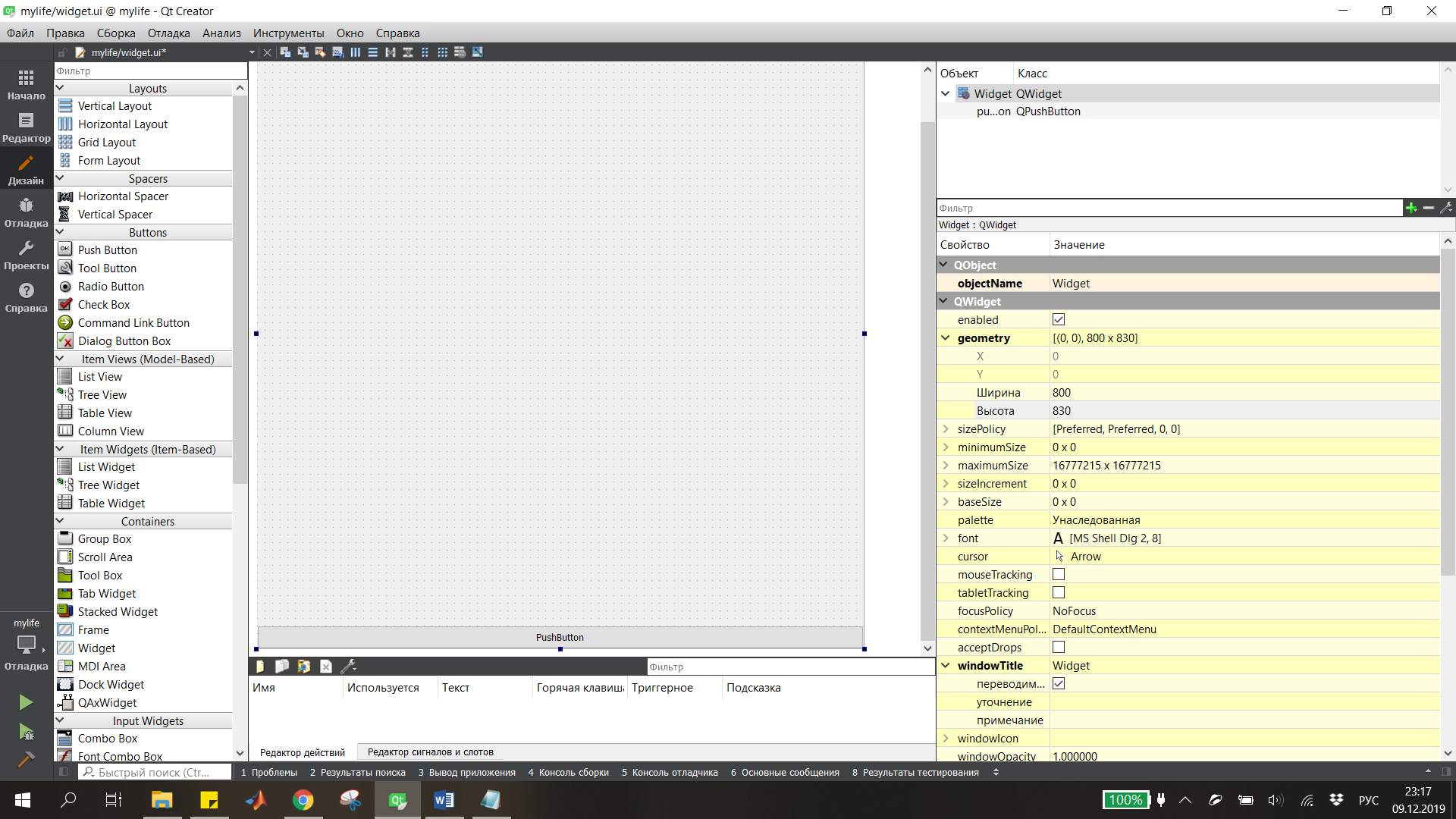
else

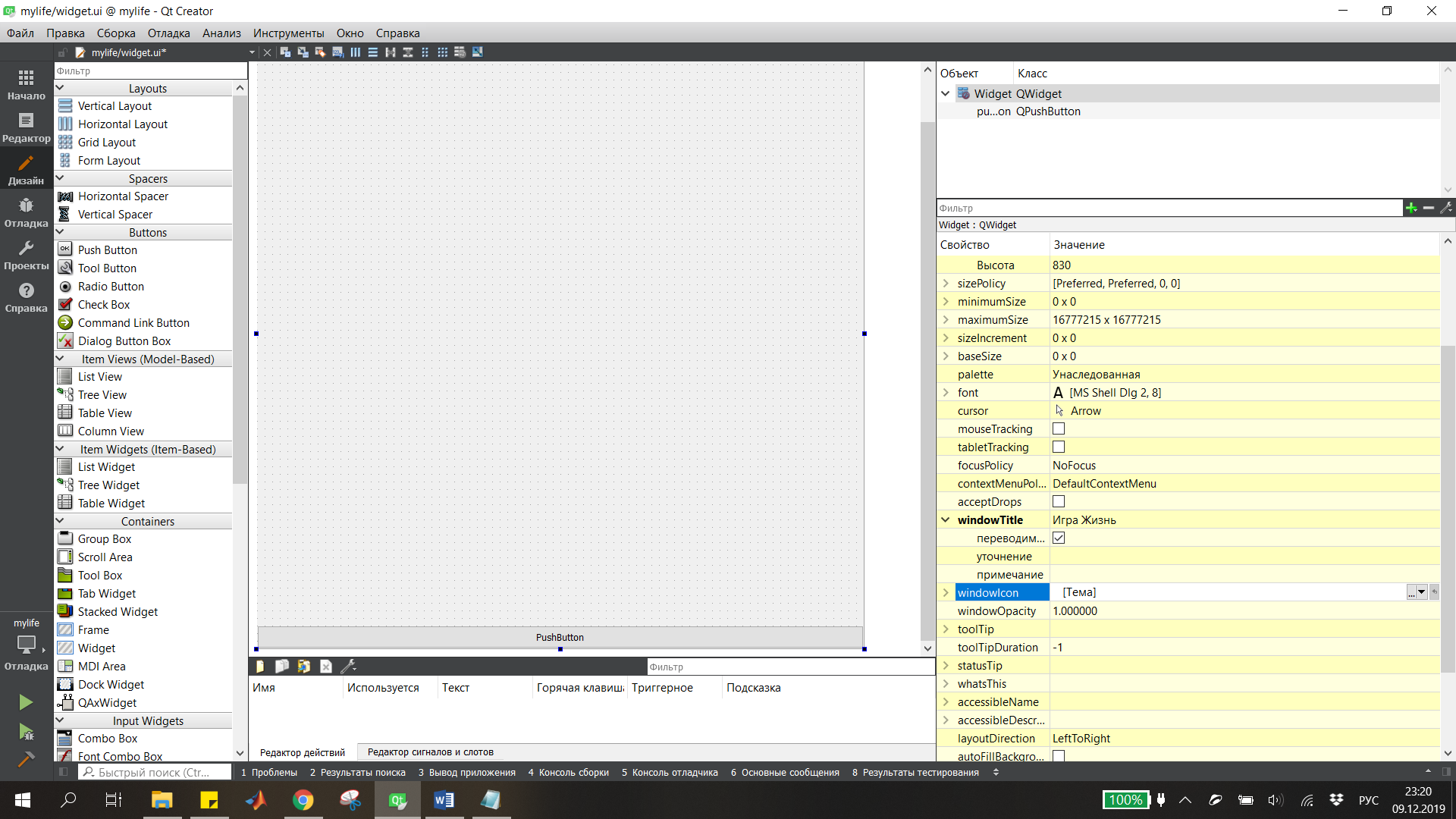
this->setBrush(QColor(255,255,255));

QGraphicsRectItem::mousePressEvent(event);

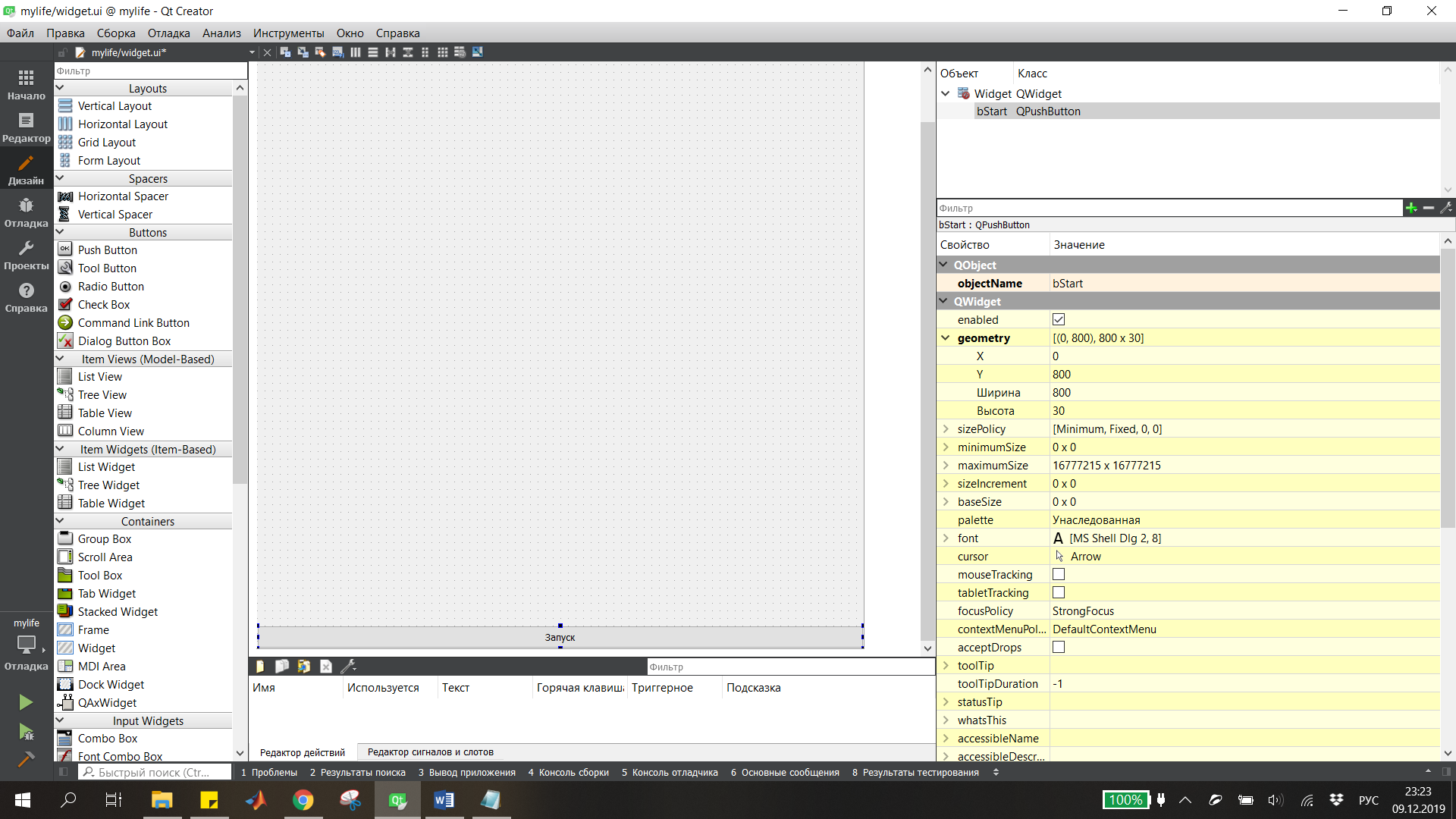
}

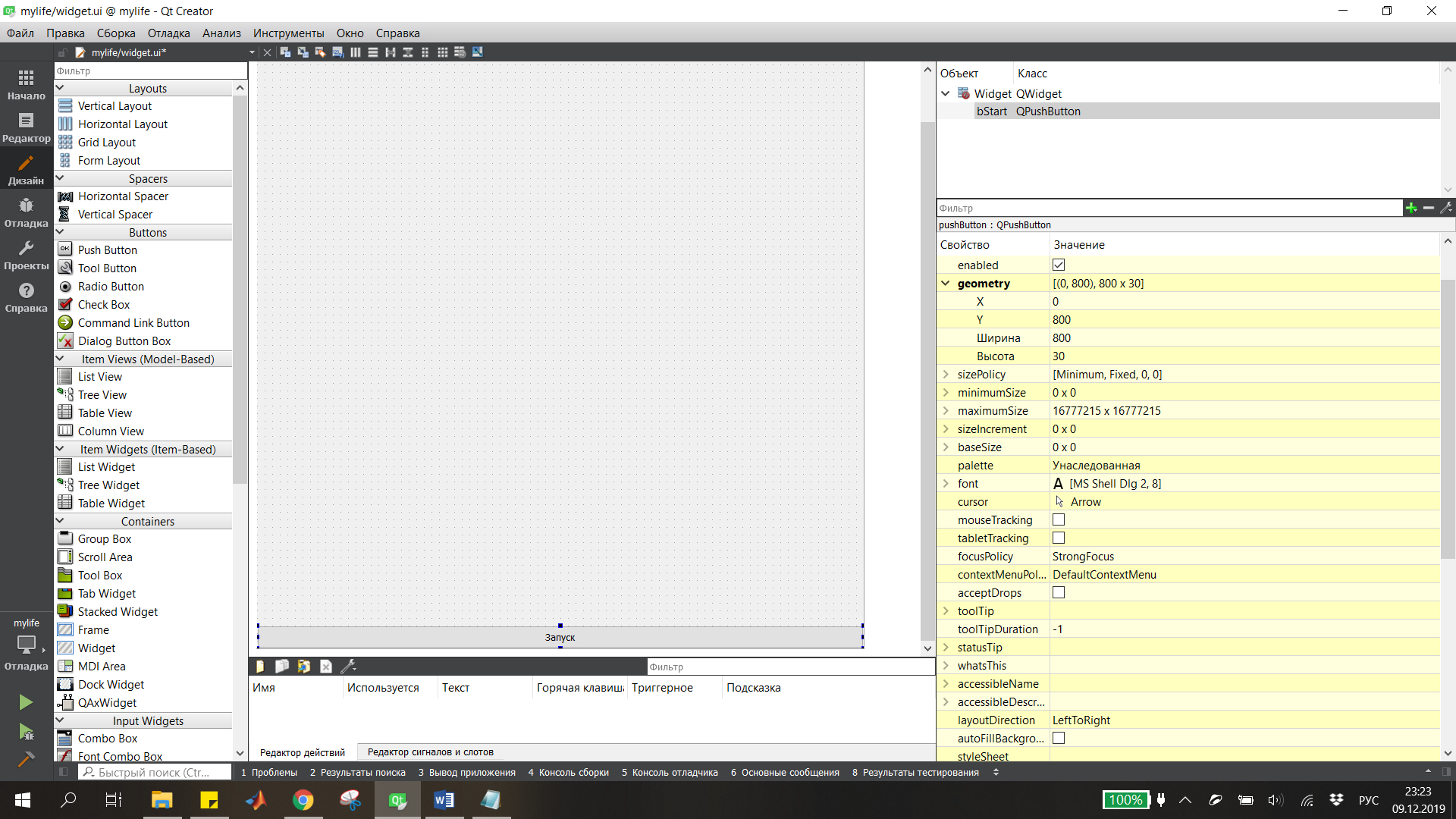
1. Далее создадим на основе QWidget графическое окно – окно игры. В окне будут располагаться два графических компонента: кнопка Запуск (начать игру) и виджет QGraphicsView (для отображения содержимого QGraphicsScene, которая может содержать текст и графические объекты). Для этого находим файлик widget.ui и два раза щелкаем по нему. Открывается режим дизайна. Настраиваем размер Widget.

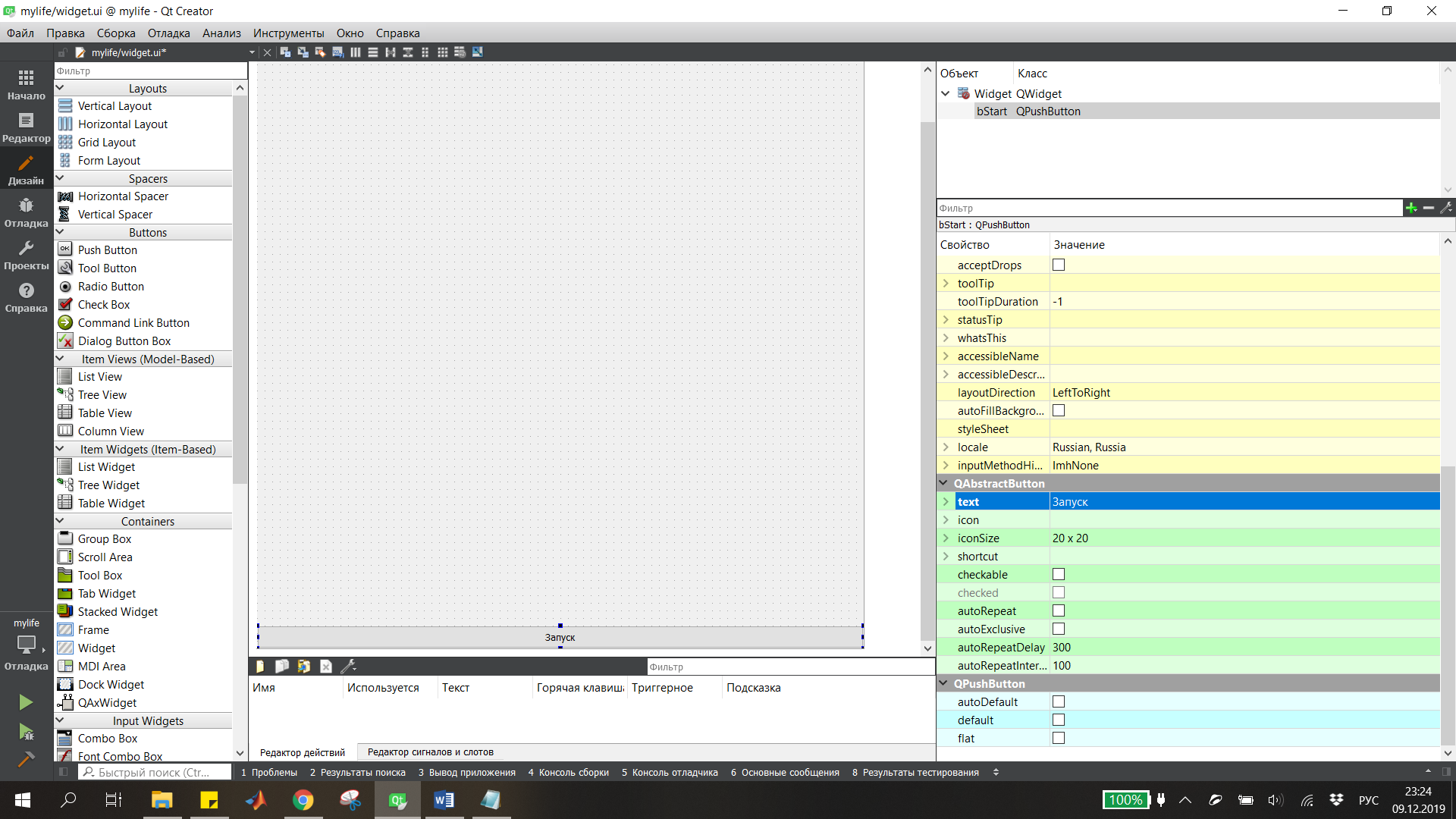




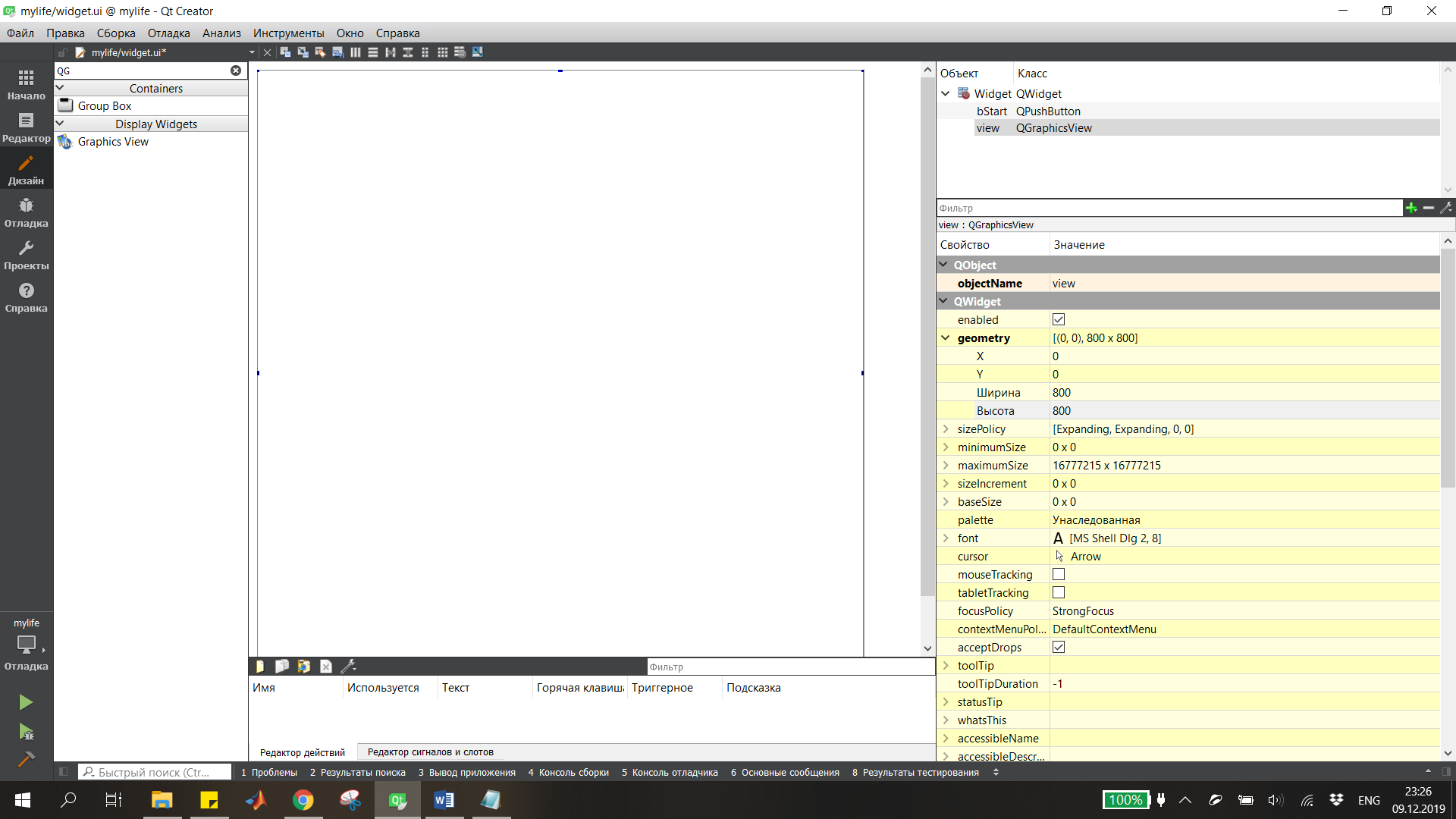
Помещаем на форму объект QPushButton и настраиваем его.







Помещаем на форму объект QGraphicsView и настраиваем его.



1. После создания QGraphicsView создаем QGraphicsScene с размерами, точно вписывающимися в QGraphicsView, чтобы не было прокруток. N – количество клеток по вертикали, M – количество клеток по горизонтали, cells – список (вектор) объектов Cell, т.е. наших клеточек. Далее создаем клетки и добавляем их в созданную QGraphicsScene методом addItem(item), и наконец загружаем QGraphicsScene в QGraphicsView.

widget.h

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include "cell.h"

#define N 50

#define M 50

namespace **Ui** {

class **Widget**;

}

class **Widget** : public QWidget

{

Q\_OBJECT

public:

explicit **Widget**(QWidget \*parent = nullptr);

~***Widget***();

void ***showEvent***(QShowEvent \* event);

private:

Ui::Widget \*ui;

Cell \*pole[N][M];

QGraphicsScene \*scene;

};

#endif

widget.cpp

void Widget::***showEvent***(QShowEvent \*event)

{

scene = new QGraphicsScene(0,0,ui->view->viewport()->width(),ui->view->viewport()->height());

qreal x = scene->sceneRect().width()/M, y = scene->sceneRect().height()/N;

for (int i=0; i<M; i++)

for (int j=0; j<N; j++)

{

pole[i][j] = new Cell(i\*x,j\*y,x,y);

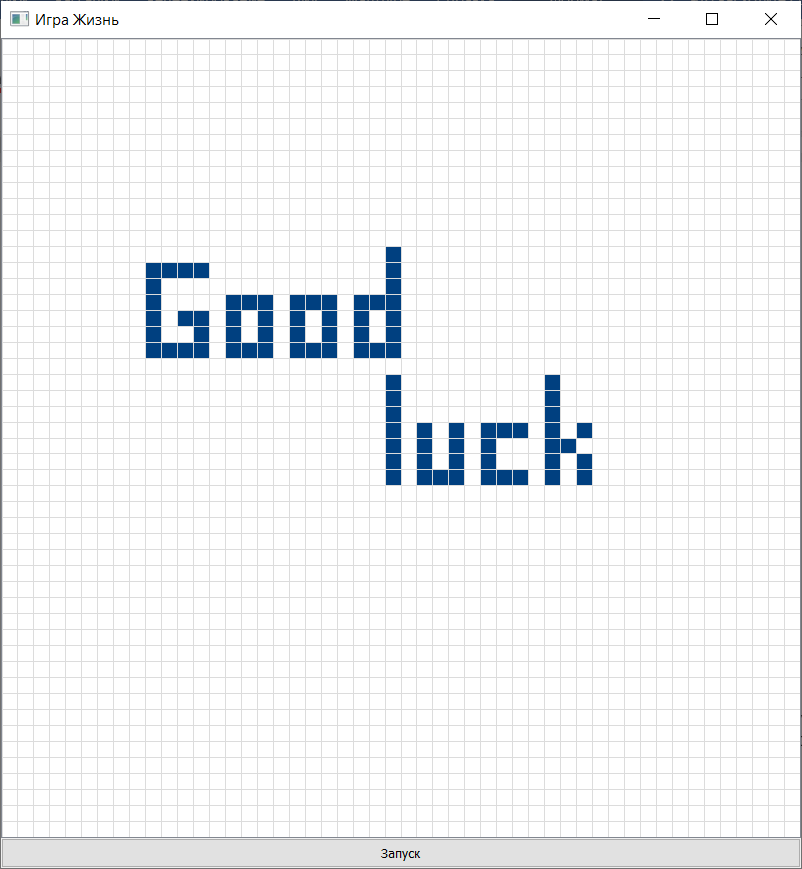
scene->addItem(pole[i][j]);

}

ui->view->setScene(scene);

}

1. Запускаем! Сборка -> Запуск. Можем кликать по квадратикам, они будут закрашиваться, а если кликать правой кнопкой, то стираться. Попробуйте! Так будем задавать первое поколение клеток.



1. Пришло время оживить игру. Для этого нам понадобится объект класса QTimer (предоставляет регулярные таймеры, которые по истечении заданного промежутка времени посылают сигнал timeout()). У нас интервал 500 мс.

widget.h

class **Widget** : public QWidget

{

Q\_OBJECT

public:

explicit **Widget**(QWidget \*parent = nullptr);

~***Widget***();

void ***showEvent***(QShowEvent \* event);

private:

Ui::Widget \*ui;

Cell \*pole[N][M];

QGraphicsScene \*scene;

QTimer \*timer;

};

widget.cpp

Widget::**Widget**(QWidget \*parent) : QWidget(parent), ui(new Ui::Widget)

{

ui->setupUi(this);

timer = new QTimer();

timer->setInterval(500);

}

1. Интервал задается в мс, т.е. таймер будет посылать сигналы каждые 500 мс (0.5 с). Добавьте в класс Widget метод-слот ontimer, который будет срабатывать после каждого сигнала таймера, т.е. 2 раза в секунду. В методе создается пустой список redraw – это координаты клеток, которые необходимо перерисовать (оживить или убить). Количество «живых» соседей определяется методом count(i,j), который мы пока не описали, о нем будет рассказано ниже. После того, как все клетки, требующие перерисовки определены, можно их перерисовать методом changeColor().

widget.h

class **Widget** : public QWidget

{

Q\_OBJECT

public:

explicit **Widget**(QWidget \*parent = nullptr);

~***Widget***();

void ***showEvent***(QShowEvent \* event);

int **count**(int i,int j);

private slots:

void **ontimer**();

private:

Ui::Widget \*ui;

Cell \*pole[N][M];

QGraphicsScene \*scene;

QTimer \*timer;

};

widget.cpp

void Widget::**ontimer**()

{

QVector<Cell\*> redraw;

for (int i=0; i<N; i++)

for (int j=0; j<N; j++)

{

int nebs = count(i,j);

if (pole[i][j]->brush().color() != QColor(255,255,255) && (nebs<2 || nebs>3))

redraw.push\_back(pole[i][j]);

if (pole[i][j]->brush().color() == QColor(255,255,255) && nebs==3)

redraw.push\_back(pole[i][j]);

}

for (int i=0; i<redraw.length(); i++)

redraw[i]->changeColor();

}

1. Для того, чтобы сигнал таймера приводил к вызову метода ontimer, присоединим сигнал timeout к слоту ontimer.

widget.cpp

Widget::**Widget**(QWidget \*parent) : QWidget(parent), ui(new Ui::Widget)

{

ui->setupUi(this);

timer = new QTimer();

timer->setInterval(500);

connect(timer,SIGNAL(timeout()),this,SLOT(ontimer()));

}

1. Разберемся с методом count(i,j), который должен подсчитывать «живых» соседей для каждой клетки. Мы будем реализовывать замкнутое поле, т.е. у клетки с координатами (0,0) будут следующие соседи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (0,0) | (0,1) |  |  |  | (0,M-1) |
| (1,0) | (1,1) |  |  |  | (1,M-1) |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| (N-1,0) | (N-1,1) |  |  |  | (N-1,M-1) |

Это должно быть учтено в методе count(i,j).

widget.h

class **Widget** : public QWidget

{

Q\_OBJECT

public:

explicit **Widget**(QWidget \*parent = nullptr);

~***Widget***();

void ***showEvent***(QShowEvent \* event);

void **ontimer**();

int **count**(int i,int j)

private:

**Ui**::**Widget** \***ui**;

Cell \*pole[N][M];

QGraphicsScene \*scene;

QTimer \*timer;

};

widget.cpp

int Widget::**count**(int i,int j)

{

int t,b,l,r;

t = i == 0 ? N-1 : i-1;

b = i == N-1 ? 0 : i+1;

l = j == 0 ? N-1 : j-1;

r = j == N-1 ? 0 : j+1;

int nebs = 0;

if (pole[t][l]->brush().color() != QColor(255,255,255))

nebs++;

if (pole[t][j]->brush().color() != QColor(255,255,255))

nebs++;

if (pole[t][r]->brush().color() != QColor(255,255,255))

nebs++;

if (pole[b][l]->brush().color() != QColor(255,255,255))

nebs++;

if (pole[b][j]->brush().color() != QColor(255,255,255))

nebs++;

if (pole[b][r]->brush().color() != QColor(255,255,255))

nebs++;

if (pole[i][l]->brush().color() != QColor(255,255,255))

nebs++;

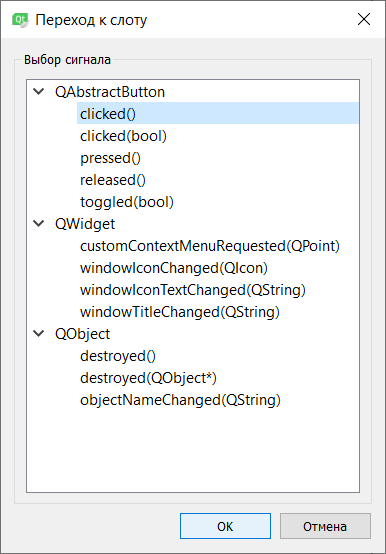
if (pole[i][r]->brush().color() != QColor(255,255,255))

nebs++;

return nebs;

}

1. Наконец, по нажатию кнопки нужно запустить таймер. Создадим метод-слот для нашей кнопки запуск. Заходим в дизайнер формы. Щелкаем по кнопке правой кнопкой мыши -> Перейти к слоту… -> OK. Создается новый метод-слот.



widget.h

class **Widget** : public QWidget

{

Q\_OBJECT

public:

explicit **Widget**(QWidget \*parent = nullptr);

~***Widget***();

void ***showEvent***(QShowEvent \* event);

int **count**(int i,int j);

private slots:

void **on\_bStart\_clicked**();

void **ontimer**();

private:

Ui::Widget \*ui;

Cell \*pole[N][M];

QGraphicsScene \*scene;

QTimer \*timer;

};

widget.cpp

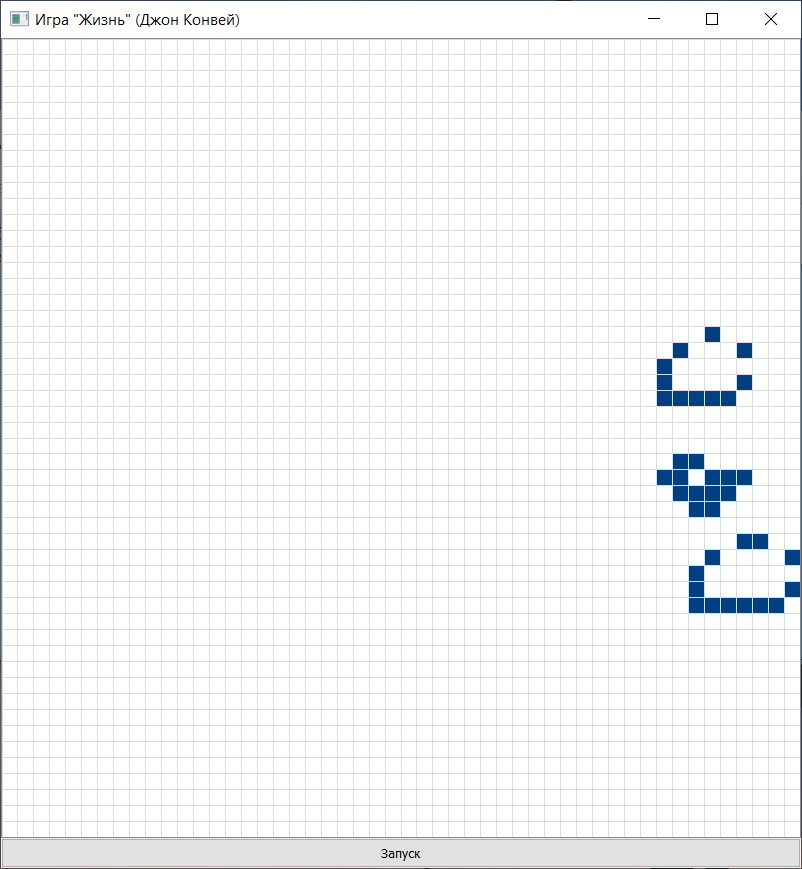
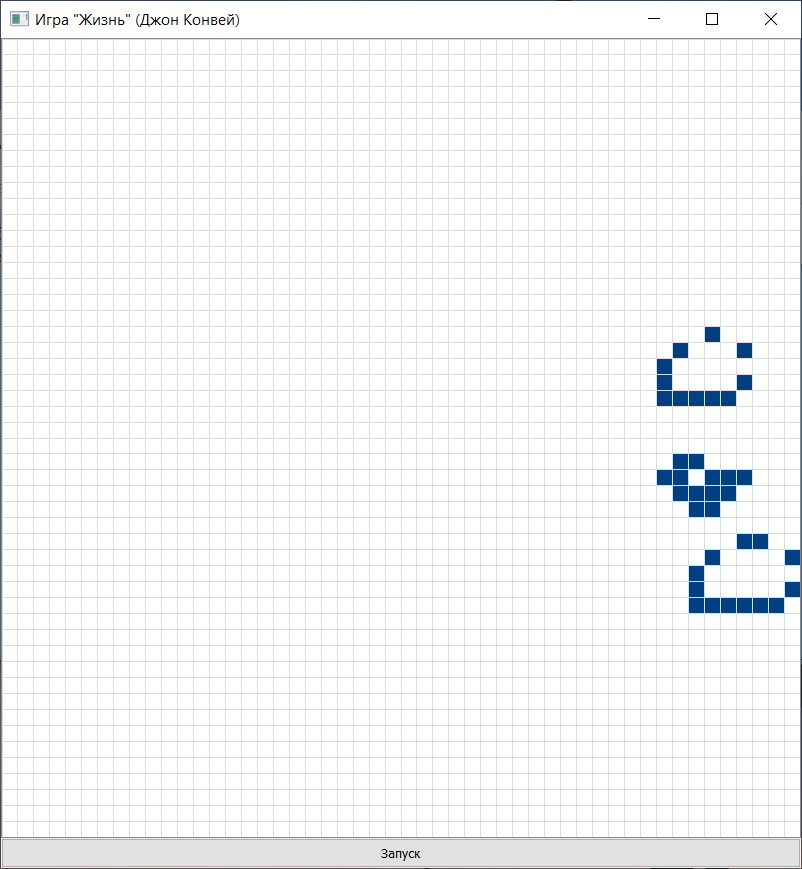
void Widget:: **on\_bStart\_clicked** ()

{

timer->start();

}

1. Теперь можно поиграть, давайте запустим паровоз с мигалками. Вам не показалось, игру активно изучали и нашли кучу интересных структур. Начальная конфигурация паровоза выглядит так (не ошибитесь, главное – точность :-)

Задание. Придумайте и реализуйте для игры Жизнь свой дизайн, пусть, например, сетка будет белая, «живые» клетки – желтые, а «мертвые» – черные. Увеличьте или уменьшите размер поля в клеточках. Добавьте кнопки установки игры на паузу, сброса игры, создания случайного начального состояния. Вперед!